BugMaps-Granger: инструмент для визуализации и предсказания ошибок с использованием тестов на причинно-следственную связь Грейнджера

Аннотация

Введение

Несмотря на увеличивающееся количество инструментов анализа ошибок для исследования ошибок в программных системах, не существует инструментов, поддерживающих исследование причинно-следственных отношений между внутренними метриками качества и ошибками. В этой статье мы предлагаем расширение инструмента BugMaps под названием BugMaps-Granger, которое позволяет анализировать свойства исходного кода, которые более вероятно приводят к ошибкам. Для этой цели мы полагаемся на тест Грейнджера для оценки того, можно ли использовать прошлые изменения в данном временном ряду метрик исходного кода для прогнозирования изменений во временном ряде дефектов. Наш инструмент извлекает версии исходного кода из платформ управления версиями, вычисляет временные ряды метрик исходного кода и дефектов, вычисляет результаты теста Грейнджера и предоставляет интерактивные визуализации для анализа причинности ошибок.

Результаты

Мы приводим пример использования BugMaps-Granger на данных систем Equinox Framework и Eclipse JDT Core, собранных за три года. Для этих систем инструмент смог идентифицировать модули с большим количеством ошибок, среднюю продолжительность и сложность ошибок, а также свойства исходного кода, которые более вероятно приводят к ошибкам.

Выводы

С результатами, предоставленными инструментом, сопровождающий может выполнять по меньшей мере две основные задачи по обеспечению качества программного обеспечения: (a) рефакторинг свойств исходного кода, вызывающих ошибки, и (б) улучшение покрытия модульными тестами в классах с большим количеством ошибок.

1 Введение

Было предложено множество инструментов анализа программного обеспечения для улучшения его качества. Такие инструменты используют различные типы информации о структуре и истории программных систем. В целом, они используются для анализа эволюции программного обеспечения, управления качеством исходного кода, вычисления метрик, проверки правил кодирования и т. д. В общем, такие инструменты помогают сопровождающим понять большие объемы данных, поступающих из хранилищ программного обеспечения.

В частности, растет интерес к инструментам анализа ошибок в программных системах. Такие инструменты помогают сопровождающим понять распределение, эволюционное поведение, продолжительность и стабильность ошибок. Например, Churrasco - это веб-инструмент для совместного анализа эволюции программного обеспечения. Цель заключается в том, чтобы предоставить расширяемый инструмент, который можно использовать для рассмотрения эволюции программного обеспечения с различных точек зрения, включая поведение ошибок. Другие визуализации также были предложены для понимания поведения ошибок, включая системную радиографию (которая предоставляет высокоуровневую визуализацию частей системы, больше всего затронутых ошибками) и часы ошибок (которые полагаются на метафору часов для предоставления информации о конкретной ошибке). Hatari - это инструмент, который предоставляет представления для просмотра наиболее рискованных мест и анализа истории риска определенного компонента из системы. Более недавно инструмент in\*Bug был предложен для позволяет пользователям навигировать и проверять информацию, хранящуюся в платформах отслеживания ошибок, с конкретной целью поддержки понимания отчетов об ошибках.

Несмотря на увеличение числа инструментов анализа ошибок, они обычно не предоставляют механизмов для оценки наличия корреляций между внутренним качеством программной системы и возникновением ошибок. На наш взгляд, не существует инструментов анализа ошибок, которые выделяют возможные причины ошибок в исходном коде. Более конкретно, не существует инструментов, предназначенных для вывода возможных причинно-следственных связей между изменениями значений метрик исходного кода и возникновением

дефектов в объектно-ориентированных классах.

В данной статье мы предлагаем и описываем инструмент BugMaps-Granger - расширение инструмента BugMaps (Hora et al. 2012), поддерживающее обнаружение причинных связей между метриками исходного кода и ошибками. Инструмент предоставляет механизмы для извлечения данных из репозиториев программного обеспечения, вычисления метрик исходного кода, генерации временных рядов метрик и дефектов и вывода причинных связей между свойствами исходного кода и дефектами. Кроме того, BugMaps-Granger предоставляет визуализации для идентификации модулей с большим количеством ошибок, средней продолжительности и сложности ошибок, а также свойств исходного кода, которые более вероятно приводят к ошибкам. Более конкретно, наш инструмент полагается на тест причинности Грейнджера (Грейнджер 1981) для идентификации причинных связей между временными рядами метрик исходного кода и дефектов. Этот тест оценивает, можно ли использовать прошлые изменения в данном временном ряду метрик исходного кода для прогнозирования изменений во временном ряде дефектов. Предложенный инструмент имеет следующие функции:

• Инструмент автоматически извлекает модели исходного кода целевой системы из ее платформы управления версиями в предопределенные временные интервалы.

• Инструмент генерирует временные ряды двенадцати метрик исходного кода и временные ряды с количеством дефектов в каждом классе целевой системы.

• Инструмент вычисляет тест Грейнджера, учитывая временные ряды метрик и дефектов, чтобы выявить возможные причинные связи.

• Инструмент интегрирует извлеченные из исходного кода модели с моделями, представляющими количество ошибок.

• Инструмент предоставляет набор интерактивных визуализаций для поддержки сопровождающих в ответах на вопросы, такие как: (а) Какие модули имеют больше ошибок? (б) Какова средняя продолжительность ошибок? (в) Какова сложность ошибок? (г) Какие свойства исходного кода вызывают ошибки в данном модуле?, и (д) Какие метрики имеют наибольшее количество положительных тестов Грейнджера?

Конечная цель BugMaps-Granger - предсказывать изменения в исходном коде, которые более вероятно приведут к дефектам. Например, имея наш инструмент под рукой, сопровождающий (перед совершением коммита с изменениями в данном классе) может проверить, повлияют ли такие изменения на значения метрик исходного кода, которые в прошлом вызывали дефекты. Если изменения значительно влияют на эти значения метрик, сопровождающий может, например, провести дополнительные меры по обеспечению качества программного обеспечения (например, провести дополнительное модульное тестирование или выполнить детальный анализ кода) перед выполнением коммита.

В предыдущей конференционной статье мы описали исследовательское исследование по использованию теста Грейнджера для прогнозирования ошибок (Couto et al. 2012). Недавно эта статья была расширена с конкретным подходом, который полагается на тесты Грейнджера для срабатывания тревог всякий раз, когда применяются рискованные изменения в исходном коде (Couto et al. 2014). Предварительная версия BugMaps - без какой-либо поддержки тестов Грейнджера - описана в короткой статье (Hora et al. 2012). Позже мы предложили вторую версию этого инструмента, который мы назвали BugMaps-Granger, включая поддержку причинности Грейнджера (Couto et al. 2013a). В настоящей статье мы расширяем эту первоначальную работу по BugMaps-Granger, включая более подробное представление об инструменте и исследование случая с двумя большими открытыми системами (Eclipse JDT Core и Equinox Framework).

2 Реализация

Выполнение инструмента BugMaps-Granger разделено на две фазы: предварительную обработку и визуализацию. Фаза предварительной обработки отвечает за извлечение моделей исходного кода, создание

временных рядов и применение теста Грейнджера для вычисления возможных причинно-следственных связей между метриками исходного кода и ошибками. В фазе визуализации пользователь взаимодействует с инструментом. Например, он может получить самые дефектные классы системы и визуализировать свойства исходного кода, вызывающие ошибки в таких классах.

BugMaps-Granger реализован в Moose (Moose platform 2014), который является платформой для анализа программного обеспечения и данных (Nierstrasz et al. 2005). На рисунке 1 показана архитектура BugMaps-Granger, которая включает четыре модуля: извлечение моделей, создание временных рядов, модуль теста Грейнджера и модуль визуализации.

Рисунок 1

2.1 Извлечение моделей

Этот модуль получает на вход URL, связанный с платформой управления версиями целевой системы (SVN или Git), и временной интервал, который будет использоваться при анализе ошибок. Для извлечения моделей исходного кода модуль выполняет следующие задачи: (а) он извлекает версии исходного кода из платформ управления версиями с интервалами в две недели; (б) он удаляет тестовые классы, предполагая, что такие классы реализованы в каталогах и подкаталогах, имена которых начинаются со слов "Test" или "test"; и (в) он разбирает версии исходного кода и генерирует файлы MSE с использованием инструмента VerveineJ (Ducasse et al. 2011; VerveineJ parser 2014). MSE - это формат файла по умолчанию, поддерживаемый платформой Moose для сохранения моделей исходного кода.

2.2 Создание временных рядов

Для создания временных рядов метрик исходного кода этот модуль получает на вход модели, извлеченные предыдущим модулем. Для каждого класса каждой извлеченной модели модуль полагается на платформу Moose для вычисления одиннадцати метрик исходного кода, включая шесть метрик CK (предложенных Chidamber и Kemerer 1994) и пять других, таких как строки кода, FAN-IN, FAN-OUT и т. д. В таблице 1 показаны метрики исходного кода, рассматриваемые инструментом. На втором этапе этот модуль сохраняет вычисленные значения метрик в CSV-файлах. По сути, для целевой системы S и метрики M этот модуль создает CSV-файл, строки которого представляют классы S, а столбцы представляют двухнедельные интервалы, рассматриваемые при извлечении версий S. Ячейка (c, t) в этом файле содержит значение метрики M, измеренное для класса c, в двухнедельном интервале t.

Таблица 1 Метрики исходного кода, рассматриваемые BugMaps-Granger

Для создания временных рядов дефектов для каждого класса этот модуль получает на вход CSV-файл, содержащий ошибки (ID и даты создания), собранные с платформ отслеживания ошибок (например, Bugzilla, Jira, Mantis и т. д.). По сути, модуль сопоставляет ошибки их соответствующим коммитам, используя стратегию сопоставления, подробно описанную в (Couto et al. 2012; Couto et al. 2014). Затем файлы исходного кода, измененные такими коммитами, используются для идентификации классов, измененных для исправления соответствующих ошибок.

2.3 Модуль теста Грейнджера

Этот модуль применяет тест причинно-следственной связи Грейнджера, рассматривая временные ряды метрик и дефектов. Для применения теста Грейнджера модуль полагается на алгоритм 1. В этом алгоритме Classes - это набор всех классов системы (строка 1), а Defects [c] - это временной ряд с количеством дефектов (строка 2). Алгоритм полагается на функцию d\_check (строка 3), чтобы проверить, соответствуют ли дефекты во временном ряду d следующим предусловиям:

Алгоритм 1 Применение теста Грейнджера

• P1: Временной ряд должен иметь как минимум 30 значений. Мотивацией для этого предусловия является тот факт, что классы, существующие только в небольшой части временного интервала, рассматриваемого в анализе, не представляют собой

интереса.

• P2: Среднее значение временного ряда должно быть больше 0,01. Это предусловие учитывает тот факт, что временные ряды с низким средним значением обычно не показывают стабильные отношения с другими временными рядами.

• P3: Временной ряд должен иметь стандартное отклонение больше 0,01. Это предусловие учитывает тот факт, что временные ряды с низким стандартным отклонением обычно не показывают стабильные отношения с другими временными рядами.

Функция d\_check вернет false, если какое-либо из предыдущих предусловий не выполняется (строка 3). В противном случае функция d\_check вернет результат теста Грейнджера для временных рядов d и m (строка 4). Для целей применения этого теста в данной статье мы используем реализацию библиотеки Granger causality testing in Python.

После применения теста Грейнджера модуль сохраняет результаты в XML-файле, который содержит четыре группы результатов. Группа 1 - это результаты теста Грейнджера, для которых d\_cause\_m равно True, то есть есть значимая причинная связь между временными рядами d и m. Группа 2 - это результаты теста Грейнджера, для которых d\_cause\_m равно False, то есть нет значимой причинной связи между временными рядами d и m. Группы 3 и 4 - это те же самые результаты, что и группы 1 и 2, но для теста Грейнджера, где временной ряд m предварительно развернут. Таким образом, существует два варианта результата теста Грейнджера: d\_cause\_m и m\_cause\_d. Формат XML-файла подробно описан в (Couto et al. 2013a).

2.4 Модуль визуализации

Этот модуль реализует визуализации для анализа причинности ошибок. В настоящей версии BugMaps-Granger поддерживает следующие визуализации:

• Тепловые карты - это тепловые карты, представляющие результаты теста Грейнджера в виде прямоугольной сетки, где каждый прямоугольник представляет класс, изменение значений метрик и дефектов и результаты теста Грейнджера.

• Диаграммы весов - это диаграммы, представляющие средние веса атрибутов из моделей исходного кода для классов, где тест Грейнджера считается значимым (в соответствии с d\_cause\_m), и атрибуты сортируются по убыванию их средних весов. Это позволяет сопровождающим определить наиболее важные атрибуты для каждого класса.

• Гистограммы весов - это гистограммы, представляющие распределение весов атрибутов из моделей исходного кода для всех классов, где тест Грейнджера считается значимым (в соответствии с d\_cause\_m).

3 Оценка качества инструмента

В этом разделе мы оцениваем качество инструмента, а также предоставляем два примера использования. Для оценки качества мы измеряем точность и полноту выявления модулей, изменение значений метрик исходного кода и дефектов, а также точность и полноту выявления атрибутов исходного кода, вызывающих ошибки.

4 Примеры использования

В этом разделе мы представляем два примера использования BugMaps-Granger на двух крупных открытых системах: Equinox Framework и Eclipse JDT Core. Для каждой системы мы исследуем изменение значений метрик исходного кода и дефектов в течение трех лет, а также свойства исходного кода, вызывающие ошибки.